02180: Introduction to Artificial Intelligence

Lecture 4: Non-determinism and Partial Observability

Nina Gierasimczuk

Idealisations

So far we have only considered search problems in environments that are:

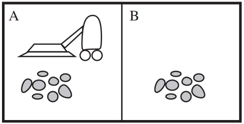
- **Single-agent**. There is a single agent acting, the one we control.
- **Static**. When the agent is not acting, the world doesn't change.
- **Deterministic**. Every action has a unique outcome.
- **Fully observable**. The full state description is accessible to the agent.

Problem solving in the real world rarely satisfies these assumptions.

Today, we will drop the assumption of **determinism** and **full observability**.

Recap: Vacuum World

Vacuum World consists of two locations, each of which may or may not contain dirt and the vacuum is in one of the locations.



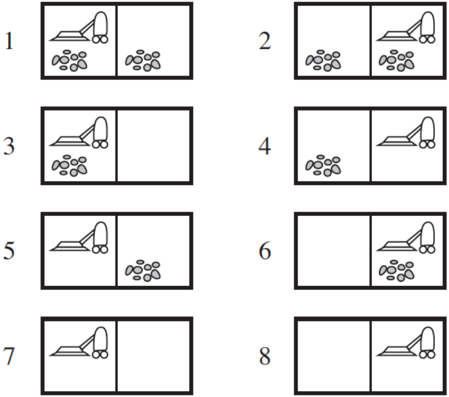
Recap: Vacuum World

States space consists of each possible configuration (2×2^2 possible states).

- s_0: Initial state
- Actions(s): for each state three possible actions: L, R, S.
- Results(s, a): applying a in s leads to a state s'.
- Goal-Test(s): *are all squares clean?*
- Step-Cost(s, a): each step costs 1.

Recap: Vacuum World

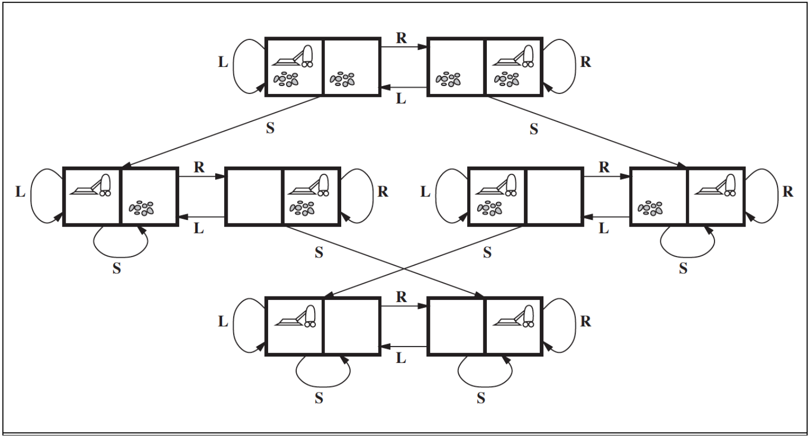
Possible states



Recap: Vacuum World

Transition Model and Simple Solution

The transitions in the Vacuum World can be represented as a graph.



Outline

Non-determinism

Partial Observability

The Erratic Vacuum World

Let us consider an erratic vacuum:

- When applied to a dirty square it cleans the square and sometimes cleans up dirt in an adjacent square, too.
- When applied to a clean square it sometimes deposits dirt in that square.

- s_0: Initial state
- ACTIONS(s): for each state three possible actions: L, R, S.
- RESULTS(s, a): applying a in s leads to **a set of states** S'.
- GOAL-TEST(s): *are all squares clean?*
- STEP-COST(s, a): each step costs 1.

- s_0: Initial state
- Actions(s): for each state three possible actions: L, R, S.
- Results(s, a): applying a in s leads to **a set of states** S'.
- Goal-Test(s): *are all squares clean?*
- Step-Cost(s, a): each step costs 1.

Q: What is a solution to the erratic vacuum problem starting in the state 1?

Representing nondeterminism

Search problems with nondeterminism: $\textsc{Results}(s, a)$ returns a *set* of states.

Example. $\textsc{Results}(1, \mathit{Suck}) = \{5, 7\}$.

Representing nondeterminism

Search problems with nondeterminism: $\textsc{Results}(s, a)$ returns a *set* of states.

Example. $\textsc{Results}((d, d, 1), \mathit{Suck}) = \{(c, d, 1), (c, c, 1)\}$.

Representing nondeterminism

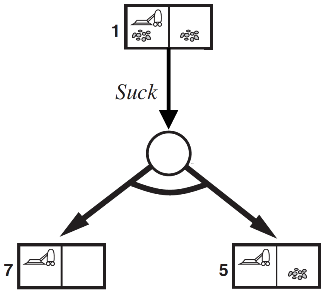
Search problems with nondeterminism: $\textsc{Results}(s, a)$ returns a *set* of states.

Example. $\textsc{Results}(1, \mathit{Suck}) = \{5, 7\}$.

Representing nondeterminism

Search problems with nondeterminism: $\textsc{Results}(s, a)$ returns a *set* of states.

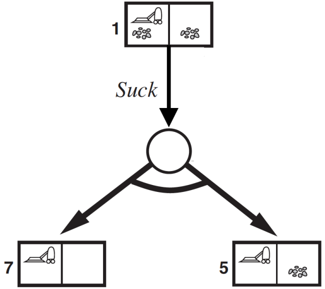
Example. $\textsc{Results}(1, \mathit{Suck}) = \{5, 7\}$.



Representing nondeterminism

Search problems with nondeterminism: $\textsc{Results}(s, a)$ returns a *set* of states.

Example. $\textsc{Results}(1, \mathit{Suck}) = \{5, 7\}$.

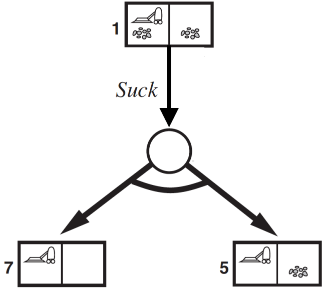


Non-deterministic action is represented with an **AND node**:
outgoing edges representing all outcomes **linked by an arc**.

Representing nondeterminism

Search problems with nondeterminism: $\textsc{Results}(s, a)$ returns a *set* of states.

Example. $\textsc{Results}(1, \mathit{Suck}) = \{5, 7\}$.



Non-deterministic action is represented with an **AND node**: outgoing edges representing all outcomes **linked by an arc**.

OR nodes are as usual, with deterministic $\textsc{Actions}$.

And-or search trees

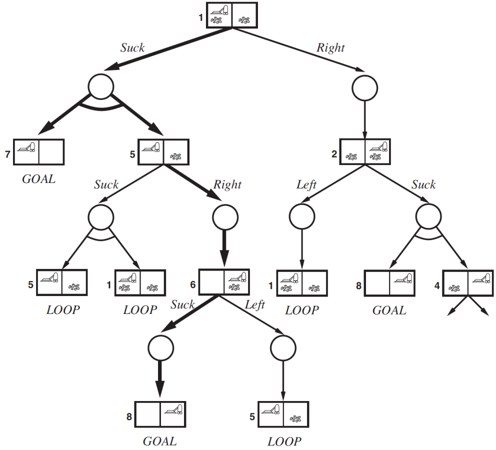
AND-OR tree is a tree with levels of AND and OR nodes.

- Root is OR node (agent choice).
- OR nodes have outgoing edges for all applicable actions.
- AND nodes have outgoing edges for all outcomes of action.

AND-OR SEARCH TREES

AND-OR tree is a tree with levels of AND and OR nodes.

- Root is OR node (agent choice).
- OR nodes have outgoing edges for all applicable actions.
- AND nodes have outgoing edges for all outcomes of action.



Solution to a Non-deterministic Search Problem

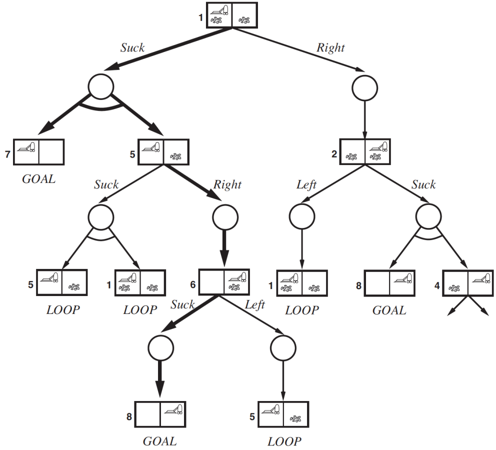
A **solution** to a nondeterministic search problem is a subtree T' of T s.t.:

1. The root note of T is in T'.
2. Every leaf of T' is a goal state.
3. Every OR node of T' has exactly one outgoing edge (agent choice).
4. Every AND node of T' has the same outgoing edges as in T (nature choice).

Solution to a Non-deterministic Search Problem

A **solution** to a nondeterministic search problem is a subtree T' of T s.t.:

1. The root note of T is in T'.
2. Every leaf of T' is a goal state.
3. Every OR node of T' has exactly one outgoing edge (agent choice).
4. Every AND node of T' has the same outgoing edges as in T (nature choice).



Solutions as conditional plans and policies

Deterministic search problems: Solution is a path in state space it can be turned into a sequence of Actions (sequential plan).

Solutions as conditional plans and policies

Deterministic search problems: Solution is a path in state space
it can be turned into a sequence of Actions (sequential plan).

Non-deterministic search problems: Solution is a subtree of state space,
it can be turned into a conditional plan (contingency plan).

Solutions as conditional plans and policies

Deterministic search problems: Solution is a path in state space
it can be turned into a sequence of $\textsc{Actions}$ (sequential plan).

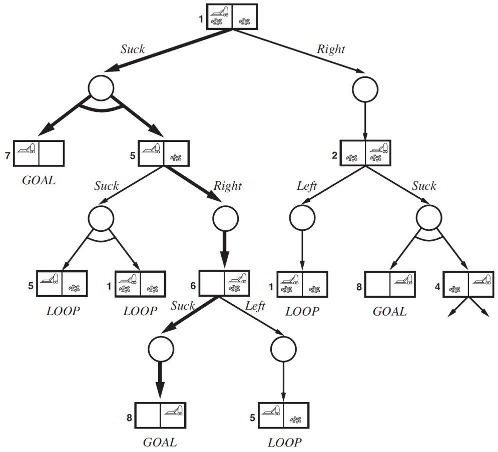
Non-deterministic search problems: Solution is a subtree of state space,
it can be turned into a conditional plan (contingency plan).

Language of **conditional plans**:

$$\pi ::= \varepsilon \mid a \mid \text{ if } s \text{ then } \pi_1 \text{ else } \pi_2 \mid \pi_1; \pi_2$$

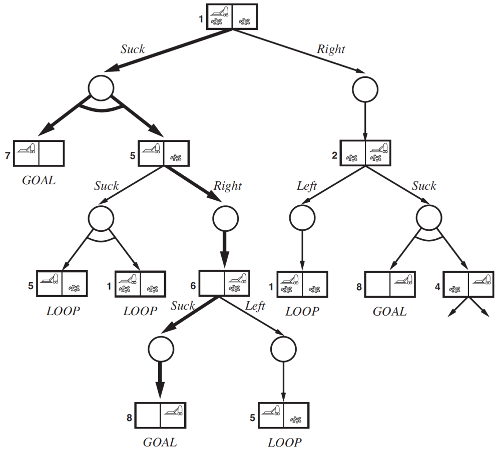
where ε is the empty plan, $a \in \textsc{Actions}$ and s is a state.
The construct $\pi_1; \pi_2$ denotes sequential composition: first execute π_1, then π_2.

Solutions as conditional plans and policies



T' as a **conditional plan**: $\pi = Suck$; if s_5 then ($Right$; $Suck$).

Solutions as conditional plans and policies



T' as a **conditional plan**: $\pi = \mathit{Suck}$; if s_5 then $(\mathit{Right}; \mathit{Suck})$.

T' as a **policy** (a mapping from states to ACTIONS):
Policy Π: $\Pi(s_1) = \mathit{Suck}$, $\Pi(s_5) = \mathit{Right}$, $\Pi(s_6) = \mathit{Suck}$.

AND-OR GRAPH SEARCH

function AND-OR-GRAPH-SEARCH(*problem*) returns a conditional plan, or failure
  OR-SEARCH(*init state of problem*, [])   // problem is implicit parameter

function OR-SEARCH(*state*, *path*)
  if *state* is a goal **then return** ε // if in goal state, empty plan suffices
  if *state* is on *path* **then return** *failure*   // fail if looping
  for each *action* applicable in *state* **do**   // recursively search for plan
    plan $\leftarrow$ AND-SEARCH(RESULTS(*state*, *action*), [*state* | *path*])
    if *plan* $\neq$ *failure* **then return** *action*; *plan*   // append plan to action
  return *failure*   // if all recursive searches for a plan fails

function AND-SEARCH(*states*, *path*)
  for each s_i **in** *states* **do**   // recursively find plans for each outcome state
    $plan_i \leftarrow$ OR-SEARCH(s_i, *path*)
    if $plan_i$ = *failure* **then return** *failure*
  return if s_1 then $plan_1$ else if s_2 then $plan_2$ else $\cdots$ if s_{n-1} then $plan_{n-1}$ else
      $plan_n$ // if $n = 1$ then the returned plan is just $plan_1$

AND-OR GRAPH SEARCH

function AND-OR-GRAPH-SEARCH(*problem*) returns a conditional plan, or failure
  OR-SEARCH(*init state of problem*, [])   // problem is implicit parameter

function OR-SEARCH(*state*, *path*)
  if *state* is a goal **then return** ε // if in goal state, empty plan suffices
  if *state* is on *path* **then return** *failure*   // fail if looping
  for each *action* applicable in *state* **do**   // recursively search for plan
    plan $\leftarrow$ AND-SEARCH(RESULTS(*state*, *action*), [*state* | *path*])
    if *plan* $\neq$ *failure* **then return** *action*; *plan*   // append plan to action
  return *failure*   // if all recursive searches for a plan fails

function AND-SEARCH(*states*, *path*)
  for each s_i **in** *states* **do**   // recursively find plans for each outcome state
    $plan_i \leftarrow$ OR-SEARCH(s_i, *path*)
    if $plan_i$ = *failure* **then return** *failure*
  return if s_1 then $plan_1$ else if s_2 then $plan_2$ else $\cdots$ if s_{n-1} then $plan_{n-1}$ else
      $plan_n$ // if $n = 1$ then the returned plan is just $plan_1$

Q: Which algorithm do we get when all ACTIONS are deterministic?

The Slippery Vacuum World

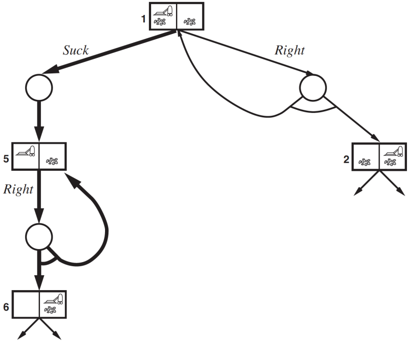
Let us consider a slipping vacuum:

- everything normal, except that
- movement action sometimes fails.

The Slippery Vacuum World

Let us consider a slipping vacuum:

- everything normal, except that
- movement action sometimes fails.



Acyclic and cyclic solutions

Acyclic solutions are not always possible.

Acyclic and cyclic solutions

Acyclic solutions are not always possible.

Cyclic solutions: Subtrees T' where every leaf is either a goal or a loop node.

Cyclic solutions guarantee reaching the goal, but only when assuming **fairness**:
Every possible outcome of an action will eventually happen.

Acyclic and cyclic solutions

Acyclic solutions are not always possible.

Cyclic solutions: Subtrees T' where every leaf is either a goal or a loop node.

Cyclic solutions guarantee reaching the goal, but only when assuming **fairness**: Every possible outcome of an action will eventually happen.

We need a language of conditional plans with loops.

$$\pi ::= \varepsilon \mid a \mid \text{ if } s \text{ then } \pi_1 \text{ else } \pi_2 \mid \pi_1 ; \pi_2 \mid \text{ while } \mathit{cond} \text{ do } \pi_1$$

where *cond* is some logical condition.

Acyclic and cyclic solutions

Acyclic solutions are not always possible.

Cyclic solutions: Subtrees T' where every leaf is either a goal or a loop node.

Cyclic solutions guarantee reaching the goal, but only when assuming **fairness**:
Every possible outcome of an action will eventually happen.

We need a language of conditional plans with loops.

$$\pi ::= \varepsilon \mid a \mid \text{ if } s \text{ then } \pi_1 \text{ else } \pi_2 \mid \pi_1;\pi_2 \mid \text{ while } \mathit{cond} \text{ do } \pi_1$$

where *cond* is some logical condition.

The slipping vacuum: *Suck* ; *Right*; while *not in* 6 do *Right*; *Suck*

Outline

Non-determinism

Partial Observability

Limiting Observability

We are now back to deterministic domains.

Limiting Observability

We are now back to deterministic domains.

We considered searching under **full observability**:
the full state description is observable by the agent.

Limiting Observability

We are now back to deterministic domains.

We considered searching under **full observability**:
the full state description is observable by the agent.

Under **partial observability** the full state might not be observable:
e.g. vacuum cleaning robot not knowing which squares are clean.

Limiting Observability

We are now back to deterministic domains.

We considered searching under **full observability**:
the full state description is observable by the agent.

Under **partial observability** the full state might not be observable:
e.g. vacuum cleaning robot not knowing which squares are clean.

Null observability: is the case when *nothing* can be observed about the states.
Search problems under null observability are called
conformant problems or **sensorless problems**.

Limiting Observability

We are now back to deterministic domains.

We considered searching under **full observability**:
the full state description is observable by the agent.

Under **partial observability** the full state might not be observable:
e.g. vacuum cleaning robot not knowing which squares are clean.

Null observability: is the case when *nothing* can be observed about the states.
Search problems under null observability are called
conformant problems or **sensorless problems**.

Example. Consider the vacuum where it is not known which squares are clean,
and the robot doesn't have any sensors.

Q: Can the problem still be solved, and if so, what is the solution?

Belief states

Belief state: A set of (physical) states.
Contains the states considered possible by an agent in a given state.

Belief states

Belief state: A set of (physical) states.
Contains the states considered possible by an agent in a given state.

We generally use s for (physical) states and b for belief states.

Belief states

Belief state: A set of (physical) states.
Contains the states considered possible by an agent in a given state.

We generally use s for (physical) states and b for belief states.

If a problem has n states, there can be up to 2^n belief states:
an exponential blow-up in the worst case.

Belief states

Belief state: A set of (physical) states.
Contains the states considered possible by an agent in a given state.

We generally use s for (physical) states and b for belief states.

If a problem has n states, there can be up to 2^n belief states:
an exponential blow-up in the worst case.

Conformant problems can be solved by
any of the standard graph and tree search algorithms (e.g. $A^\star$),
just using belief states instead of physical states.

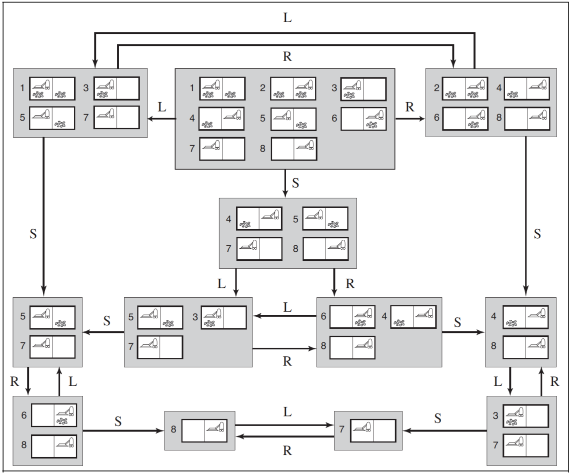
Belief states

Formally, given a fully observable problem
$(s_0, \textsc{Actions}, \textsc{Results}, \textsc{Goal-Test})$,
we can define a corresponding conformant problem
$(b_0, \textsc{Actions}', \textsc{Results}', \textsc{Goal-Test}')$
with initial belief state b_0 by:

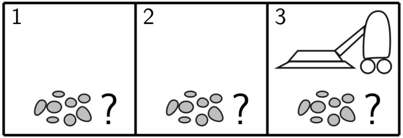
$$\begin{aligned}
\textsc{Actions}'(b) &= \bigcup_{s \in b} \textsc{Actions}(s) \\
\textsc{Results}'(b, a) &= \bigcup_{s \in b} \textsc{Results}(s, a) \\
\textsc{Goal-Test}(b) &= \bigwedge_{s \in b} \textsc{Goal-Test}(s)
\end{aligned}$$

Conformant Search

Deterministic Sensorless Vacuum



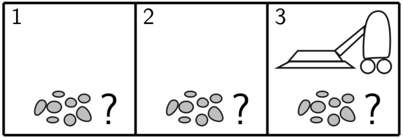
Searching with Observations (Sensing)



Assume the robot doesn't initially know which squares are dirty,
but it has a *Sense* action to check whether the current square is clean or dirty.

Suppose the number of *Suck* actions have to be minimised.

Searching with Observations (Sensing)

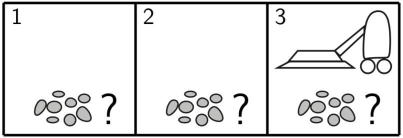


Assume the robot doesn't initially know which squares are dirty,
but it has a *Sense* action to check whether the current square is clean or dirty.

Suppose the number of *Suck* actions have to be minimised.

Q1: What would then be a solution to the cleaning problem?

Searching with Observations (Sensing)



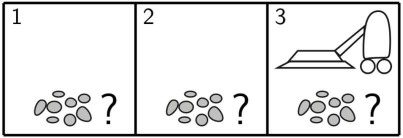
Assume the robot doesn't initially know which squares are dirty, but it has a *Sense* action to check whether the current square is clean or dirty.

Suppose the number of *Suck* actions have to be minimised.

Q1: What would then be a solution to the cleaning problem?

Q2: Can we use the Graph-Search to solve problems with partial observability and sensing?

Searching with Observations (Sensing)



Assume the robot doesn't initially know which squares are dirty, but it has a *Sense* action to check whether the current square is clean or dirty.

Suppose the number of *Suck* actions have to be minimised.

Q1: What would then be a solution to the cleaning problem?

Q2: Can we use the Graph-Search to solve problems with partial observability and sensing?

Q3: And what about And-Or-Graph-Search?

Percepts as a Model for Sensing

A general treatment of observations/sensing under partial observability:

including a new function in the problem description: $\textsc{Percept}(s)$ which codes the observation made by the agent in state s.

Percepts as a Model for Sensing

A general treatment of observations/sensing under partial observability:

including a new function in the problem description: $\textsc{Percept}(s)$ which codes the observation made by the agent in state s.

Example: $\textsc{Percept}(s) = \mathit{dirty}/\mathit{clean}$.

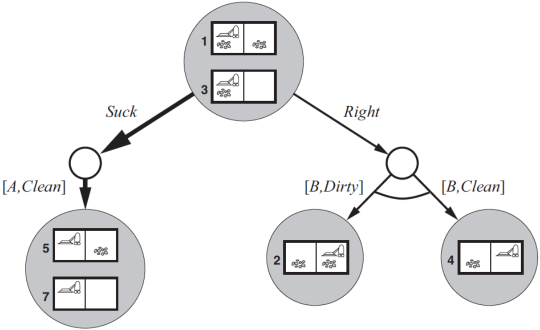
Percepts as a Model for Sensing

A general treatment of observations/sensing under partial observability:

including a new function in the problem description: $\text{PERCEPT}(s)$ which codes the observation made by the agent in state s.

Example: $\text{PERCEPT}(s) = \textit{dirty}/\textit{clean}$.

Full observability corresponds to $\text{PERCEPT}(s) = \{s\}$.
Null observability corresponds to $\text{PERCEPT}(s) = \emptyset$.



The new RESULTS function

We define the following new functions:

1. $\textsc{Possible-Percepts}$ takes a belief state b and returns all the observations that are possible to receive in that belief state:

$$\textsc{Possible-Percepts}(b) = \{\textsc{Percept}(s) \mid s \in b\}.$$

2. $\textsc{Update}$ takes a belief state b and an observation o and filters away the states that are not consistent with the observation:

$$\textsc{Update}(b, o) = \{s \in b \mid o = \textsc{Percept}(s)\}.$$

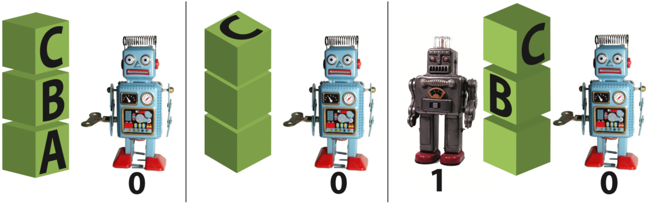
   So $\textsc{Update}(b, o)$ is the updated belief an agent has after having received observation o in belief state b.

New $\textsc{Results}$ function on belief states that takes observations into account:

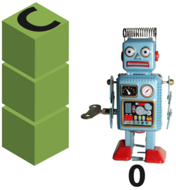
$$\begin{aligned}\textsc{Results}'(b, a) = \quad & \{\textsc{Update}(\textstyle\bigcup_{s \in b} \textsc{Results}(s, a), o) \mid \\ & o \in \textsc{Possible-Percepts}(\textstyle\bigcup_{s \in b} \textsc{Results}(s, a))\}\end{aligned}$$

Note that $\textsc{Results}(b, a)$ is a set of belief states, that is, a set of sets of states.

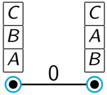
From Full to Partial Observability to Multiple Agents

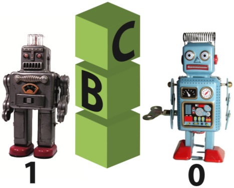


From Full to Partial Observability to Multiple Agents

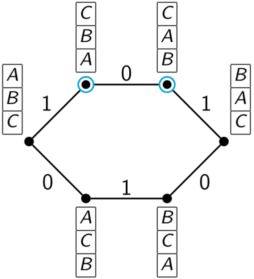


$\curvearrowright$





$\curvearrowright$



Such models are one of the main topics of the course 02287.

THE END OF LECTURE 4